AMENDMENTS TO THE CLAIMS

1. (previously presented, allowed) A method, comprising the steps of:

decoding a macroinstruction of a computer, the decoding of the macroinstruction generating a plurality of iterations of a sequence of one or more microinstructions, including:

a pattern of microinstructions implementing a basic operation, and

a branch instruction predicted not taken;

on detecting that an iteration completes operation of the macroinstruction, adding a marker indicating the end of the macroinstruction to a microinstruction in the pipeline downstream of the instruction decoder;

after reaching a termination condition of the macroinstruction, partially executing an iteration beyond the termination, the partial execution committing at least one side-effect to an architecturally-visible resource of the computer, and raising an exception to transfer control to a second microinstruction stream; and

in a second microinstruction stream, unwinding the side-effects committed by the post-termination iteration.

2. (cancelled)

3. (previously presented, allowable) The method of claim 5,

on detecting that an iteration completes operation of the macroinstruction, adding a marker indicating the end of the macroinstruction to a microinstruction in the pipeline downstream of the instruction decoder.

4. (previously presented, allowable) The method of claim 5, further comprising the steps of:

partially executing a loop iteration beyond the termination condition of a loop of a decoded microinstruction stream, the partial execution committing at least one side-effect to an architecturally-visible resource of the computer;

raising an exception to transfer control to a second microinstruction stream;

in the second microinstruction stream, unwinding the side-effects committed by the post-termination iteration.

5. (previously presented) A method comprising the steps of:

decoding a macroinstruction of a computer, the decoding of the macroinstruction generating a plurality of iterations of:

a pattern of microinstructions implementing a basic operation, wherein the microinstruction set is architecturally exposed to programs fetched from an architecturally-visible memory of the computer, and

a branch instruction predicted not taken.

6. (original) The method of claim 5, wherein instructions of the microinstruction set are managed by a memory management unit between a main memory of the computer and one or more cache levels.

7. (cancelled)

8. (currently amended) The computer of claim 13, further comprising:

instruction fetch and execution circuitry designed to fetch and execute instructions in both the macroinstruction set and the microinstruction set, each of the two instruction sets to include ~~including~~ store instructions to write data to a memory of the computer;

store monitoring circuitry designed to monitor the store instructions and to invalidate any copies of a datum in memory overwritten by the store instructions, including copies of instructions in any instruction cache in the instruction set other than the instruction set of the current store instruction.

9. (previously presented, last amended 3/1/2006, allowable) The computer of claim 13, further comprising:

execution circuitry designed to execute an instruction calling for waiting to allow a pipeline to drain and to set bits of a floating-point control word to values denoted in an explicit immediate field of the instruction.

10. (cancelled)

11. (cancelled)

12. (currently amended, last amended 3/1/2006) The computer of claim 13, wherein the branch microinstruction is a branch instruction available to a program to be fetched from a memory of the computer.

13. (currently amended) A computer, comprising:

an instruction decoder designed to decode macroinstructions into microinstructions for execution in an instruction pipeline on a computer, and for at least one macroinstruction that includes internal iterations, the decoding of the internal-iteration macroinstruction designed to generate ~~generating~~ a plurality of iterations of:

a pattern of microinstructions for implementing a basic operation of an internal iteration of the internal-iteration macroinstruction, and

a branch microinstruction predicted not taken, wherein the branch microinstruction is to be generated carrying a marker indicating that the branch microinstruction defines a boundary between two successive iterations of the internal-iteration macroinstruction;

the instruction decoder being further designed to cease generating iterations on detection of a branch mispredict.

14. (previously presented, allowed) A method comprising the steps of:

decoding a macroinstruction on a computer, the macroinstruction calling for a plurality of iterations of a sequence of one or more microinstructions; and

on detecting that an iteration completes operation of the macroinstruction, adding a marker indicating the end of the macroinstruction to a microinstruction in the pipeline downstream of the instruction decoder.

15. (original, allowed) The method of claim 14, further comprising the steps of:

after reaching a termination condition of the iterations, partially executing an iteration beyond the termination, the partial execution committing at least one side-effect to an architecturally-visible resource of the computer, and raising an exception to transfer control to a second microinstruction stream;

in the second microinstruction stream, unwinding the side-effects committed by the post-termination iteration.

16. (original, allowed) The method of claim 14, further comprising the steps of:

executing a microinstruction on the computer, the microinstruction storing into a memory location a value of a second instruction coded in the macroinstruction set;

in response to the storing, clearing a memory system, including an instruction cache, and execution pipeline of the computer of the former content of the memory location;

executing the second instruction in the execution pipeline.

17. (original, allowed) The method of claim 14, wherein the detection of termination comprises detecting a branch mispredict.

18. (previously presented, allowed) The method of claim 17, wherein the mispredict is detected on a branch microinstruction architecturally available to a program fetched from a memory of the computer.

19. (original, allowed) The method of claim 14, wherein the branch microinstruction is generated carrying a marker indicating that the branch microinstruction defines a boundary between two successive iterations.

20. (previously presented, allowed) A computer, comprising:

an instruction decoder designed to decode a macroinstruction set that includes a macroinstruction calling for a plurality of iterations of a sequence of one or more microinstructions;

a pipeline stage downstream of the instruction decoder designed to detect that operation of the macroinstruction is complete, and in response, to add to a microinstruction a marker indicating the end of the macroinstruction.

21. (previously presented, allowed) The computer of claim 20:

wherein the decoder, when decoding the macroinstruction, is designed to generate a plurality of iterations of:

a pattern of microinstructions implementing a basic operation, and

a branch microinstruction predicted not taken.

22. (original, allowed) The computer of claim 20,

circuitry designed to execute an instruction calling for waiting to allow a pipeline to drain and to set bits of a floating-point control word to values denoted in an explicit immediate field of the instruction.

23. (original, allowed) A method, comprising the steps of:

after reaching a termination condition of a loop of a first microinstruction stream executing in a computer, the microinstruction stream being generated by decoding a macroinstruction, partially executing a loop iteration beyond the termination, the partial execution committing at least one side-effect to an architecturally-visible resource of the computer, and raising an exception to transfer control to a second microinstruction stream;

in the second microinstruction stream, unwinding the side-effects committed by the post-termination iteration.

24. (original, allowed) The method of claim 23,

executing a microinstruction of a computer, the microinstruction storing into a memory location a value of a second instruction coded in a second instruction set;

in response to the storing, clearing an instruction cache and execution pipeline of the computer of the former content of the memory location;

executing the second instruction in the execution pipeline.

25. (original, allowed) The method of claim 23, further comprising the step of:

executing an instruction that calls for waiting to allow a pipeline to drain and setting bits of a floating-point control word to values denoted in an explicit immediate field of the instruction.

26. (original, allowed) The method of claim 23, further comprising the step of:

ceasing to generate iterations when a termination condition of the macroinstruction is detected in an execution stage of an instruction pipeline of the computer.

27. (original, allowed) The method of claim 26, wherein the detection of termination comprises detecting a branch mispredict.

28. (original, allowed) The method of claim 23, wherein each iteration completes with a branch microinstruction architecturally available to a program fetched from a memory of the computer.

29. (original, allowed) The method of claim 28, wherein the branch microinstruction is generated carrying a marker indicating that the branch microinstruction defines a boundary between two successive iterations.

30. (currently amended, allowed) A computer, comprising:

circuitry designed to partially execute a post-termination iteration of a loop of a first microinstruction stream executing in the computer, the partial execution designed to commit ~~committing~~ at least one side-effect to an architecturally-visible resource of the computer, and to raise an exception to transfer control to a second microinstruction stream;

software of the second microinstruction stream, programmed to unwind side-effects committed by the post-termination iteration.

31. (previously presented, allowed) The computer of claim 30,

an instruction decoder designed to decode macroinstructions into microinstructions for execution in an instruction pipeline on a computer, and for at least one macroinstruction, the decoding of the macroinstruction generating a plurality of iterations of:

a pattern of microinstructions implementing a basic operation, and

a branch microinstruction predicted not taken.

32. (original, allowed) The computer of claim 30, further comprising:

a pipeline stage downstream of an issue buffer of the computer, designed to detect the termination condition of the loop, and in response, to add to a microinstruction a marker indicating the end of the macroinstruction.

33. (original, allowed) The computer of claim 30, wherein the loop of the first microinstruction stream consists essentially of instructions generated by decoding a single macroinstruction fetched from memory of the computer.

34. (original, allowed) The computer of claim 30, wherein an instruction generator of the computer joins iterations of the loop by branches predicted not taken.

35. (original, allowed) A method comprising the steps of:

in a computer having instruction fetch circuitry for fetching instructions in first and second instruction sets from a memory of the computer and executing the instructions, executing a first instruction coded in the first instruction set, the first instruction storing into a memory location a value of a second instruction coded in the second instruction set,

in response to the storing, clearing an instruction cache and execution pipeline of the computer of the former content of the memory location;

executing the second instruction in the execution pipeline.

36. (previously presented, allowed) The method of claim 35,

decoding an instruction of the second instruction set, the decoding generating a plurality of iterations of:

a pattern of instructions of the first instruction set implementing a basic operation, and

a branch instruction of the first instruction predicted not taken.

37. (original, allowed) The method of claim 35, further comprising the step of:

executing an instruction calling for waiting to allow a floating-point pipeline of the computer to drain and to set bits of a floating-point control word to values denoted in an explicit immediate field of the instruction.

38. (original, allowed) The method of claim 35, an instruction decoder for the second instruction set designed to generate instructions in the first instruction set for execution in the execution pipeline.

39. (original, allowed) The method of claim 35, wherein the instructions in the execution pipeline are not tagged with an indication of an instruction set of origin.

40. (original, allowed) The method of claim 35, wherein the monitoring is based on comparing addresses in a physical address space.

41. (original, allowed) A computer, comprising:

instruction fetch and execution circuitry designed to fetch and execute instructions in two different instruction sets, each instruction set including store instructions to write data to a memory of the computer;

store monitoring circuitry designed to monitor the store instructions and to invalidate any copies of a datum overwritten by the store instructions, including copies of instructions in any instruction cache, in the instruction set other than the instruction set of the current store instruction.

42. (previously presented, allowed) The computer of claim 41,

an instruction decoder designed to decode an instruction in a first of the instruction sets, and in response, to generate a plurality of iterations of a sequence of one or more instructions in a second one of the instruction sets;

a pipeline stage downstream of the instruction decoder designed to detect that operation of the decoded instruction is complete, and in response, to add to a one of the generated instructions a marker indicating the end of the macroinstruction.

43. (currently amended, allowed) The computer of claim 41:

wherein the instruction fetch and execute circuitry is further designed to partially execute a post-termination iteration of a loop within a first instruction in a first one of the instruction sets, the partial execution to commit ~~committing~~ at least one side-effect to an architecturally-visible resource of the computer, and to raise an exception to transfer control to a second instruction stream in a second one of the instruction sets, programmed to unwind side-effects committed by the post-termination iteration.

44. (original, allowed) A method, comprising the steps of:

decoding and executing an instruction on a computer, execution of the instruction comprising the steps of waiting to allow a pipeline to drain, and setting bits of a floating-point control word to values denoted in an explicit immediate field of the instruction.

45. (original, allowed) The method of claim 44, wherein instruction fetch and execution circuitry of the computer are designed to fetch and execute a macroinstruction set and a microinstruction set from memory.

46. (previously presented, allowed) The method of claim 45,

decoding a macroinstruction of the computer, the decoding of the macroinstruction generating a plurality of iterations of:

a pattern of microinstructions implementing a basic operation, and

a branch instruction predicted not taken.

47. (previously presented, allowed) The method of claim 45, further comprising the steps of:

decoding a macroinstruction on the computer, the macroinstruction calling for a plurality of iterations of a sequence of one or more microinstructions; and

on detecting that an iteration completes operation of the macroinstruction, adding a marker indicating the end of the macroinstruction to a microinstruction in the pipeline downstream of the instruction decoder.

48. (original, allowed) The method of claim 45, further comprising the step of:

emitting the instruction as a microinstruction in response to a macroinstruction whose execution is dependent on a full/empty state of a floating-point top-of-stack.

49. (original, allowed) A computer, comprising:

execution circuitry designed to execute an instruction calling for waiting to allow a pipeline to drain and to set bits of a floating-point control word to values denoted in an explicit immediate field of the instruction.

50. (original, allowed) The computer of claim 49, wherein instruction fetch and execution circuitry of the computer are designed to fetch and execute instructions in two different instruction sets.

51. (previously presented, last amended 3/1/2006, allowed) The computer of claim 49, further comprising:

each instruction set including instructions defined to write data to a memory of the computer;

store monitoring circuitry designed to monitor the data write instructions and to invalidate any copies of a datum in memory overwritten by the data write instructions, including copies of instructions in the instruction set other than the instruction set of the current data write instruction.

52. (original, allowed) The computer of claim 49, the instruction specifying individual bits of the floating-point control word to be written, in addition to values to be written to those bits.

53. (original, allowed) The computer of claim 49,

the execution circuitry being further designed to execute an instruction calling for waiting to allow a pipeline to drain and to raise an exception based on a test of bits of a floating-point control word.

54. (previously presented, allowable) The method of claim 5, wherein the branch microinstruction is generated carrying a marker indicating that the branch microinstruction defines a boundary between two successive iterations.

55. (previously presented, last amended 3/1/2006, allowed) A computer, comprising:

an instruction decoder designed to decode macroinstructions into microinstructions for execution in an instruction pipeline on a computer, and for at least one macroinstruction, the decoder being designed to generate a plurality of iterations of a pattern of microinstructions implementing a basic operation of the macroinstruction, a microinstruction of each of the plurality of iteration patterns carrying a marker indicating that the marked microinstruction defines a boundary between two successive iterations, the microinstruction set being architecturally exposed for execution from an architecturally-exposed memory; and

operand commit circuitry designed to detect the marker, and in response, to commit results of an iteration to architectural state of the computer.

56. (previously presented, last amended 3/1/2006, allowed) The computer of claim 55, further comprising:

a pipeline stage downstream of the instruction decoder designed to detect that operation of the macroinstruction is complete, and in response, to add to a microinstruction a marker indicating the end of the macroinstruction.

57. (previously presented, last amended 3/1/2006, allowed) The computer of claim 55, wherein:

the microinstruction to be generated carrying ~~bearing~~ the iteration boundary marker is a branch microinstruction predicted not taken.

58. (previously presented, last amended 3/1/2006, allowed) The computer of claim 55, further comprising:

circuitry designed to partially execute a post-termination iteration of a loop of a first microinstruction stream executing in the computer, the partial execution to commit ~~committing~~

at least one side-effect to an architecturally-visible resource of the computer, and to raise an exception to transfer control to a second microinstruction stream;

software of the second microinstruction stream, programmed to unwind side-effects committed by the post-termination iteration.